IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s)	:	Madsen, et al.
Serial No.	:	09/813,522
Filing Date	:	March 21, 2001
Title	:	Dynamic Software Code Instrumentation Method and System
Group Art Unit	:	2124
Examiner	:	Q. Nahar

Mail Stop: Appeal Brief-Patent
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Certificate of Mailing
I hereby certify that this correspondence is being deposited with U.S. Postal Services as first class mail in an envelope addressed to:
Mail Stop: Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

By: _____ Date: September 7, 2005
Michael J. Marcin, (Reg. No. 48,198)

TRANSMITTAL

In response to the Notice of Appeal filed June 8, 2005 and the Advisory Action dated July 14, 2005, transmitted herewith please find three copies of an Appeal Brief for filing in the above-identified application. Applicants hereby request a one-month extension. Please charge the Credit card of Fay Kaplun & Marcin, LLP in the amount of $620.00. The Commissioner is hereby authorized to charge the **Deposit Account of Fay Kaplun & Marcin, LLP NO. 50-1492** for additional required fees. A copy of the paper is enclosed for that purpose.

Respectfully submitted,

Dated: September 7, 2005

By: _____
Michael J. Marcin, Reg. No. 48,198

Fay Kaplun & Marcin, LLP
150 Broadway, Suite 702
New York, New York 10038
Tel: (212) 619-6000
Fax: (212) 619-0276

AF
DW

PATENT
Attorney Docket No.: 40101 - 10901

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:	)	
	)	
Kenneth E. Madsen et al.	)	
	)	
Serial No.: 09/813,522	)	Group Art Unit: 2191
	)	
Filed: March 21, 2001	)	Examiner: Qamrun Nahar
	)	
For: DYNAMIC SOFTWARE CODE INSTRUMENTATION METHOD AND SYSTEM	)	**Board of Patent Appeals and Interferences**

Mail Stop: Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF UNDER 37 C.F.R. § 41.37

In support of the Notice of Appeal filed June 8, 2005, and pursuant to 37 C.F.R. § 41.37, Appellants present in triplicate their appeal brief in the above-captioned application.

This is an appeal to the Board of Patent Appeals and Interferences from the Examiner's final rejection of claims 1-49 in the final Office Action dated February 10, 2005. The appealed claims are set forth in the attached Claims Appendix.

09/13/2005 TBESHAH1 00000001 09813522
01 FC:1402 500.00 OP

09/13/2005 TBESHAH1 00000001 09813522
02 FC:1251 120.00 OP

1. Real Party in Interest

This application is assigned to Wind River Systems, Inc., the real party in interest.

2. Related Appeals and Interferences

There are no other appeals or interferences which would directly affect, be directly affected, or have a bearing on the instant appeal.

3. Status of the Claims

Claims 1-49 have been rejected in the final Office Action. The final rejection of claims 1-49 is being appealed.

4. Status of Amendments

All amendments submitted by the Appellants have been entered.

5. Summary of Claimed Subject Matter

The present invention describes a system and method for monitoring software code as it is executed by a target CPU. (See Specification, p. 6, ll. 35-38). The system includes an instrumentation block which contains an instruction locating module coupled to an instruction replacement module, which is in turn coupled to a vector table instrumentation module. (See Specification, p. 8, ll. 84-90). The instruction locating module scans a block of program code to locate predetermined instructions, such as those used to designate a preamble and/or a postamble of a code block or function. (See Specification, p. 8, ll. 90-95). The instruction replacement

module replaces the original instructions with substitute instructions designed to generate an exception (e.g., a misalignment exception) which is of a type commonly included in an instruction set of the target CPU. (See Specification, p. 8, ll. 95-102). The CPU then branches to the exception vector table, where it executes an exception routine. (See Specification, p. 11).

The exception routine may include instructions to save the current state of a Data Cache, disable/enable the Data Cache, decode the exception and execute the original instruction, turn an Instruction Cache on/off, create entry/exit markers, restore the original state of the Data Cache, and return to the next instruction following the exception. (See Specification, pp. 10-12, Tables 1, 2). Because read and write instructions normally occur in the Data Cache, disabling the Data Cache causes the CPU to execute the instructions on an external bus which is visible to an external bus/state analyzer, and allows a pre-fetched instruction at the start of the exception routine to serve as a tag which informs an external bus/state analyzer to capture the subsequent cycle of instructions being executed. Similarly, disabling the Instruction Cache allows instruction fetches to occur on the external bus. (See Specification, pp. 12-13).

6. Grounds of Rejection to be Reviewed on Appeal

I. Whether claims 1-49 are unpatentable under 35 U.S.C. § 112, second paragraph, as being indefinite.

II. Whether claims 1-9, 11-15, 18-19, 22-44, 46, 48 and 49 are unpatentable under 35 U.S.C. § 102(b) as anticipated by U.S. Patent No. 5,689,712 to

Heisch ("the Heisch patent").

III. Whether claims 20-21, 45 and 47 are unpatentable under 35 U.S.C. § 103(a) as obvious over the Heisch patent in view of U.S. Patent No. 5,664,191 to Davidson et al. ("the Davidson patent").

7. Argument

I. The Rejection of Claims 1-49 Under 35 U.S.C. § 112, Second Paragraph, as Being Indefinite Should Be Reversed.

A. The Examiner's Rejection

In the Final Office Action, the Examiner rejected claims 1-49 under 35 U.S.C. 112, second paragraph, as being indefinite. (See 2/10/05 Office Action, p. 3). The Examiner stated that there was insufficient antecedent basis for independent claims 1 and 45-49.

B. The Amendments to Claims 1 and 45-59 are Sufficient to Overcome the 35 U.S.C. § 112, Second Paragraph Rejection.

Appellants' subsequent Amendment, filed 6/8/05, included amendments to claims 1 and 45-49 which were entered by the Examiner. It is respectfully submitted that the amendments are sufficient to overcome the rejection. Thus, Appellants respectfully request that the Board overturn the Examiner's rejection under 35 U.S.C. § 112, second paragraph, of independent claims 1 and 45-49, and all claims depending therefrom (claims 2-44).

II. The Rejection of Claims 1-9, 11-15, 18-19, 22-44, 46, 48 and 49 Under 35 U.S.C. § 102(b) as Anticipated By U.S. Patent No. 5,689,712 to Heisch Should Be Reversed.

A. The Examiner's Rejection

In the Final Office Action, the Examiner rejected claims 1-9, 11-15, 18-19, 22-44, 46, 48 and 49 Under 35 U.S.C. § 102(b) as being unpatentable over the Heisch patent. (See 2/10/05 Office Action, pp. 3-16).

The Heisch patent describes a system and a process for optimizing a program having memory references. The process includes instrumenting each of the memory references to create an instrumented program, which is executed to capture effective address trace data for each of the memory references. The memory references are then reordered to optimize the program. (See the Heisch patent, col. 2, ll. 23-33).

According to the Heisch patent, instrumentation involves patching each load and store (l/s) instruction in the program with code that calculates and stores the reference address for each l/s instruction in a trace buffer. A postprocessor initializes a pointer, p, to the start of the instrumentation code, which is located at the end of the program. The instrumentation code searches for the next l/s instruction and replaces the l/s instruction with an unconditional branch to the location corresponding to the pointer p. The postprocessor then generates instrumentation code for the l/s instruction at the location corresponding to the pointer p. The l/s instruction is appended to the end of the instrumentation code, along with an unconditional branch back to the original code. The pointer p is then updated to point to the next available instrumentation code

location following the appended unconditional branch. (See the Heisch patent, col. 3, l. 65 - col. 4, l. 30). Once the instrumented program is completed, the postprocessor analyzes the trace buffer to generate an optimal reorder list. (See the Heisch patent, col. 4, ll. 62-66).

B. The Heisch Patent Does Not Disclose Monitoring Data and Replacing the Desired Instruction With a Program Flow Change Instruction Directing Execution To a Buffer as Recited in Claims 1, 46, 48 and 49.

Independent claim 1 recites "[a] method for *monitoring data* and changing a behavior of a run time execution of software code in a target system, said method comprising: . . . replacing the desired instruction with a program flow change instruction *directing execution to a buffer.*" With reference to a prior art software analysis system utilizing tag statements in instrumented source code, the specification of the present application states that:

> when a tag statement is executed in the target system, a tag is written to a predetermined location in the address space of the target system . . . A drawback of this approach, however, is that discrete tag statements, which tend to disadvantageously increase the number of executable lines of the code, must be included within the source code. These discrete tags disadvantageously increase the size of the code in proportion to the number of functions instrumented. Moreover, although the tags may be monitored, the code continues to be executed within cache, and thus is not directly observable. (See Specification, p. 2, ll. 5-23).

Initially, it should be noted that while the invention of the present application is directed towards monitoring program execution for testing, debugging, and emulation purposes, the Heisch patent relates only to program optimization. The Heisch patent does not teach

program monitoring, but rather analyzes a program simply to locate memory references in order to rearrange the program with respect to the memory references. As would be known to one skilled in the art, a memory reference refers to a location of data, rather than the data itself. The Heisch patent is not concerned with the data itself, only the location, and thus does not teach or suggest "a method for monitoring data," as recited in claim 1.

In further contrast to claim 1, the Heisch patent teaches appending instructions to the end of the program. The present application expressly teaches against adding instrumentation code to the body of the program, and states that "the invention does not necessitate increasing the size (lines) of executable code being debugged." (See Specification, p. 20). Thus, claim 1 recites "replacing the desired instruction with a program flow change instruction directing execution to a buffer." The specification includes an example of using an exception vector table as a buffer for holding the instrumentation code. It should be noted that in the Advisory Action, the Examiner noted that the Appellants were relying on the exception vector table to distinguish the present invention from the Heisch patent, but that the exception vector table was not specifically recited in the claim. (See 7/14/05 Advisory Action, p. 2). The Appellants note that they are not relying on the fact that an example of a buffer may be an exception vector table, but rather "replacing the desired instruction with a program flow change instruction directing execution to a buffer," as is specifically recited in claim 1.

In stark contrast to the present invention, execution of the system of the Heisch patent involves branching between the original program code and the instrumentation code at the

end of the program code. There is no teaching or suggestion of directing program execution to a buffer that branches between the program and the buffer (e.g., exception vector table). Therefore, the Heisch patent neither teaches nor discloses "replacing the desired instruction with a program flow change instruction directing execution to a buffer," as recited in claim 1.

Independent claims 46, 48 and 49 recite systems, an article of manufacture, and a computer readable program code, respectively, for *monitoring data* and changing a behavior of a run time execution of software code in a target system, which include an instruction *directing execution to a buffer*. Thus, for the same reasons as described above, the Heisch patent does not teach or suggest these recitations.

Accordingly, Appellants respectfully request that the Board overturn the Examiner's rejection under 35 U.S.C. § 102(b) of independent claims 1, 46, 48 and 49, and all claims dependent therefrom (claims 2-9, 11-15, 18-19 and 22-44).

III. The Rejection of Claims 20-21, 45 and 47 Under 35 U.S.C. § 103(a) as Being Obvious Over U.S. Patent No. 5,689,712 to Heisch in view of U.S. Patent No. 5,664,191 to Davidson et al. Should Be Reversed.

A. The Examiner's Rejection

In the Final Office Action, the Examiner rejected claims 20-21, 45 and 47 Under 35 U.S.C. § 103(a) as being unpatentable over the Heisch patent in view of the Davidson patent. (See 2/10/05 Office Action, pp. 17-18).

The Davidson patent describes a method and system for determining an optimal placement order for basic blocks within a computer program to improve locality of reference and reduce the working set of the program. (See the Davidson patent, Abstract). The method includes analyzing the program to identify all basic blocks, determining how many times each basic block is executed, and reordering the basic blocks based on how many times the basic blocks were executed. Execution data for each basic block is collected by running an instrumented version of the computer program. (See the Davidson patent, col. 4, l. 60 - col. 5, l. 7).

B. The Cited Patents Do Not Disclose Monitoring Data and Replacing the Desired Instruction With a Program Flow Change Instruction Directing Execution To a Buffer as Recited in Claims 1, 45 and 47.

Claim 45 includes the same recitation relied on above to distinguish claim 1 from the Heisch patent. Specifically, claim 45 recites "[a] method for *monitoring data* and changing a behavior of a run time execution of software code in a target system, said method comprising: . . . replacing the desired instruction with a program flow change instruction *directing execution to a buffer*." Claim 47 includes the same recitation relied on above to distinguish claims 46, 48 and 49 from the Heisch patent. In particular, claim 47 recites "[a] system for *monitoring data* and changing a behavior of a run time execution of software code in a target system, said system comprising: . . . an instruction replacement module configured to replace the desired instruction with a program flow change instruction *directing execution to a buffer*."

Appellants respectfully submit that the Davidson patent is insufficient to cure the deficiencies of the Heisch patent described above with reference to claim 1. As with the Heisch patent, the Davidson patent relates to program optimization, and is concerned with determining the location and access statistics of specific code elements (i.e., basic blocks). Nowhere does the Davidson patent teach or suggest monitoring program execution for testing, debugging, or emulation purposes.

Furthermore, the Davidson patent, like the Heisch patent, teaches placing instrumentation code within the body of the program. The Davidson patent states that "Instrumentation code may be manually added to the computer program 116, or the optimizer program 114 may automatically insert a call to a library routine into each basic block when the basic block is identified." (See the Davidson patent, col. 9, ll. 46-50). Thus, it is respectfully submitted that the Davidson patent does not teach or suggest "monitoring data" and "replacing the desired instruction with a program flow change instruction directing execution to a buffer," as recited in claims 1 and 45 or "an instruction replacement module configured to replace the desired instruction with a program flow change instruction directing execution to a buffer," as recited in claim 47.

Thus, Appellants respectfully request that the Board overturn the Examiner's rejection under 35 U.S.C. § 103(a) of claims 20-21, which depend from claim 1, and claims 45 and 47.

8. Conclusions

For the reasons set forth above, Appellants respectfully request that the Board reverse the final rejections of the claims by the Examiner under 35 U.S.C. § 112, 35 U.S.C. § 102(b) and 35 U.S.C. § 103(a), and indicate that claims 1-49 are allowable.

Respectfully submitted,

Date: September 7, 2005

By: _____
Michael J. Marcin (Reg. No. 48,198)

Fay Kaplun & Marcin, LLP
150 Broadway, Suite 702
New York, NY 10038
Tel: (212) 619-6000
Fax: (212) 619-0276

CLAIMS APPENDIX

1. A method for monitoring data and changing a behavior of a run time execution of software code in a target system, said method comprising:

(a) searching a range of addresses within the software code to identify a desired instruction;

(b) replacing the desired instruction with a program flow change instruction directing execution to a buffer, wherein the program flow change instruction is configured to change the behavior relative to that of the desired instruction, of the run time execution of the software code; and

(c) inserting a routine into the buffer, the routine having an output instruction and a branch instruction branching to an address of the software code subsequent to the program flow change instruction.

2. The method of claim 1, wherein the output instruction generates output to a trace buffer.

3. The method of claim 2, wherein the trace buffer is disposed on the target system.

4. The method of claim 1, comprising storing information in an instrumentation table to undo said replacing (b) and said inserting (c).

5. The method of claim 4, wherein the instrumentation table is disposed on a host system communicably coupled to the target system.

6. The method of claim 4, wherein said storing comprises storing the desired instruction, address of the desired instruction, action to be performed by the program flow change instruction, address of the buffer, size of the routine, and an identifier associated with the action to be performed.

7. The method of claim 1, wherein the target system includes a cache and at least a portion of the software code executes externally of the cache.

8. The method of claim 1, wherein the target system includes a bus and at least a portion of the software code executes on the bus.

9. The method of claim 1, wherein said searching (a) further comprises searching for a plurality of desired instructions.

10. The method of claim 1, wherein the routine comprises a cache disabling instruction and a cache re-enabling instruction.

11. The method of claim 1, wherein said searching (a) comprises searching for a desired instruction disposed at the beginning of a program function.

12. The method of claim 11, wherein the desired instruction comprises a Move From Special Register instruction.

13. The method of claim 11, wherein said searching (a) comprises searching for an other desired instruction disposed at the ending of a program function.

14. The method of claim 13, wherein the other desired instruction comprises a Move to Special Register instruction.

15. The method of claim 1, wherein said searching (a) comprises searching for at least one desired instruction associated with data manipulation.

16. The method of claim 1, wherein said inserting (c) comprises inserting a routine having a Data

Cache-disabling instruction.

17. The method of claim 1, wherein said inserting (c) comprises inserting a routine having an Instruction Cache-disabling instruction.

18. The method of claim 1, wherein said searching (a) comprises searching for a branch instruction, and searching for the desired instruction in a portion of the software code indicated by the branch instruction, the desired instruction being disposed outside of the range of addresses identified.

19. The method of claim 1, wherein the desired instruction comprises an EABI instruction.

20. The method of claim 1, wherein the searching (a) comprises using debug information to identify the desired instruction.

21. The method of claim 20, wherein the searching (a) comprises using complier-derived debug information in a format selected from the group consisting of stabs, elf, and dwarf formats.

22. The method of claim 1, wherein the program flow change instruction comprises an instruction to read from an odd address.

23. The method of claim 22, wherein the program flow change instruction comprises an instruction to add an odd integer to an address.

24. The method of claim 23, wherein the routine has a decoding instruction to identify the odd integer and execute an instruction corresponding thereto.

25. The method of claim 1, comprising a plurality of program flow change instructions

corresponding to a plurality of user-selectable operations.

26. The method of claim 25, wherein each of said plurality of user-selectable operations is selected from the group consisting of:

indicating entry and exit of a function;

indicating entry and exit of a function and tracing execution of a function;

indicating entry and exit of a function, tracing execution of the function, and indicating entry and exit and tracing execution of other functions called by the function;

indicating Entry and Exit of a function, tracing execution of the function, and indicating Entry and Exit without tracing execution of other functions called by the function;

indicating data manipulation;

inserting patch code into a code portion;

inserting the sequence of program execution; and

indicating changes to variables.

27. The method of claim 26, wherein said inserting (c)comprises:

(i) selecting at least one output code statement to perform a selected one of said user-selectable operations;

(ii) saving a copy of the output code statement and the desired instruction;

(iii) determining the size of the output code statement, the branch instruction, the desired instruction, and restore code to restore the desired instruction;

(iv) allocating memory in the buffer of the size determined in (iii); and

(v) inserting the output code statement, the branch instruction, the desired instruction, and restore code, into the allocated memory.

28. The method of claim 27, wherein said saving (ii) comprises saving a copy of the program flow change instruction and the desired instruction in a translation table.

29. The method of claim 27, wherein said selecting (i) comprises analyzing a symbol table of the software code.

30. The method of claim 27, wherein said selecting (i) comprises calling a function selected from the group consisting of a printf or scanf function.

31. The method of claim 27, wherein the restore code comprises code to save and restore original register contexts.

32. The method of claim 26, wherein said searching (a) comprises identifying addresses in the program code that are associated with each instance of a modification of an identified variable/structure, and locating a final instruction for each instance of a modification, the final instruction being said desired instruction.

33. The method of claim 32, wherein said inserting (c) comprises:

(i) selecting at least one output code statement to transfer data to the buffer;

(ii) saving a copy of the output code statement and the desired instruction;

(iii) determining the size of the output code statement, the desired instruction, and restore code to restore the desired instruction;

(iv) allocating memory in the buffer of the size determined in (iii), and to run the trace acquisition code; and

(v) inserting the output code statement, the branch instruction, the desired instruction, and restore code, into the allocated memory.

34. The method of claim 33, wherein said allocating (iv) further comprises allocating additional memory of the size determined in (iii) for each said instance of a modification of an identified variable/structure.

35. The method of claim 34, further comprising repeating said inserting (v) for each said instance.

36. The method of claim 33, wherein said saving (ii) comprises saving a copy of the program flow change instruction and the desired instruction in a translation table.

37. The method of claim 33, wherein said selecting (i) comprises analyzing a symbol table of the software code.

38. The method of claim 33, wherein said selecting (i) comprises calling a function selected from the group consisting of a printf or scanf function.

39. The method of claim 33, wherein the restore code comprises code to save and restore original register contexts.

40. The method of claim 1, further comprising reversing said replacing (b), and inserting (c), to restore the software code.

41. The method of claim 1, wherein at least one of said searching (a), replacing (b), and inserting (c) is performed during run time execution of the software code.

42. The method of claim 41, wherein at least one of said searching (a), replacing (b), and inserting (c) is performed after the software code is compiled.

43. The method of claim 42, wherein execution of the software code is halted during performance of said at least one of said searching (a), replacing (b), and inserting (c).

44. The method of claim 1, comprising executing the software code.

45. A method for monitoring data and changing a behavior of a run time execution of software code in a target system, said method comprising:

searching a range of addresses within the software code using debug information to identify a desired instruction;

replacing the desired instruction with a program flow change instruction directing execution to a buffer, wherein the program flow change instruction is configured to change the behavior, relative to that of the desired instruction, of the run time execution of the software code; and

inserting a routine into the buffer, the routine having a branch instruction branching to an address of the software code subsequent to the program flow change instruction.

46. A system for monitoring data and changing a behavior of a run time execution of software code in a target system, said system comprising:

an instruction locating module configured to search a range of addresses within the software code to identify a desired instruction;

an instruction replacement module configured to replace the desired instruction with a program flow change instruction directing execution to a buffer, wherein the program flow change instruction is configured to change the behavior, relative to that of the desired instruction, of the run time execution of the software code; and

an instrumentation module configured to insert a routine into the buffer, the routine having an output instruction and a branch instruction branching to an address of the software code subsequent to the program flow change instruction.

47. A system for monitoring data and changing a behavior of a run time execution of software code in a target system, said system comprising:

an instruction locating module configured to search a range of addresses within the software code using debug information to identify a desired instruction;

an instruction replacement module configured to replace the desired instruction with a

program flow change instruction directing execution to a buffer, wherein the program flow change instruction is configured to change the behavior, relative to that of the desired instruction, of the run time execution of the software code; and

an instrumentation module configured to insert a routine into the buffer, the routine having a branch instruction branching to an address of the software code subsequent to the program flow change instruction.

48. An article of manufacture for monitoring data and changing a behavior of a run time execution of software code in a target system, said article of manufacture comprising:

a computer usable medium having computer readable program code embodied therein, said computer usable medium having:

computer readable program code for searching a range of addresses within the software code to identify a desired instruction before or after execution thereof;

computer readable program code for replacing the desired instruction with a program flow change instruction directing execution to a buffer, wherein the program flow change instruction is configured to change the behavior, relative to that of the desired instruction, of the run time execution of the software code; and

computer readable program code for inserting a routine into the buffer, the routine having an output instruction and a branch instruction branching to an address of the software code subsequent to the program flow change instruction.

49. Computer readable program code for monitoring data and changing a behavior of a run time execution of software code in a target system, said computer readable program code comprising:

computer readable program code for searching a range of addresses within the software code to identify a desired instruction;

computer readable program code for replacing the desired instruction with a program flow change instruction directing execution to a buffer, wherein the program flow change instruction is configured to change the behavior, relative to that of the desired instruction, of the

run time execution of the software code; and

computer readable program code for inserting a routine into the buffer, the routine having an output instruction and a branch instruction branching to an address of the software code subsequent to the program flow change instruction.

EVIDENCE APPENDIX

No evidence has been entered or relied upon in the present appeal.

RELATED PROCEEDING APPENDIX

No decisions have been rendered regarding the present appeal or any proceedings related thereto.